INTRODUCTION À GIT

Formation Mésocentre ECP
03 avril 2013

Ronan Vicquelin
Ecole Centrale Paris, Laboratoire EM2C CNRS

GIT: WHAT IS IT ?

- Version control system

  - **Save history of modifications for small/large projects:**
    - **Code sources**
    - **Reports/Papers**
    - **...**

- Others

  - **CVS**
  - **SVN**
  - **Mercurial**
  - **...**

- GIT versus CVS/SVN

  - **Distributed**
  - **Full-fledged repository**
  - **Branching/merging are fast and easy**

OUTLINE

- **Basics** (git status / add / commit)

- **Branching** (git branch / checkout / merge)

- **Sharing** (git push / fetch / pull)

CONFIGURE GIT

- Set your username and email

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "your_email@youremail.com"
```

- Activate color display

```
$ git config --global color.ui true
```

- On OS X, use FileMerge instead of diff

```
$ vi ~/git-diff-cmd.sh
```

```
#!/bin/sh
/usr/bin/opendiff "$2" "$5" -merge "$1"
```

```
$ git config --global diff.external ~/git-diff-cmd.sh
```

CREATE A GIT REPOSITORY

- Initialize a git repository

```
$ mkdir myRepo
$ cd myRepo
$ git init
```

- Clone a git repository

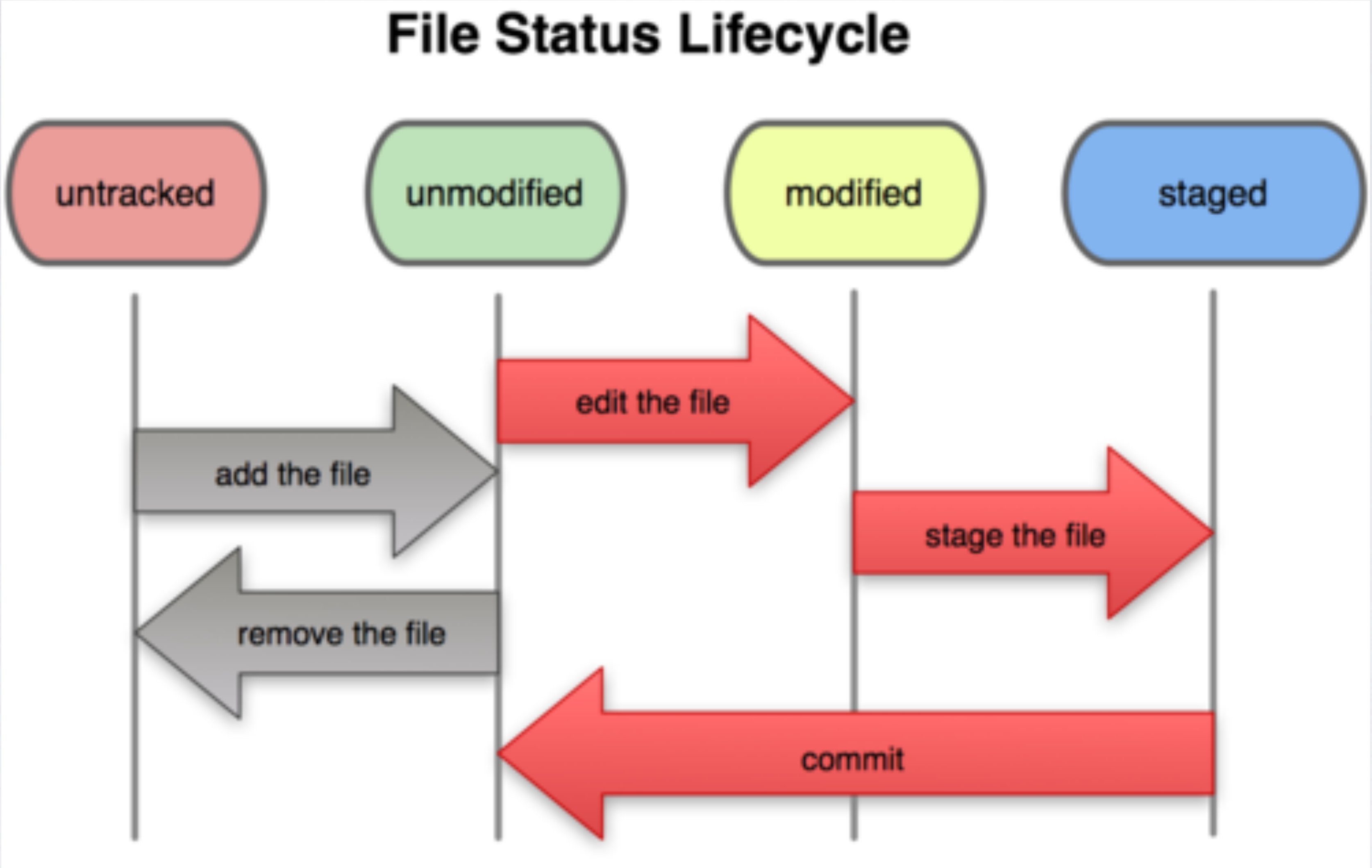
```
$ git clone git@bastion.em2c.ecp.fr:Test.git
```

```
$ git clone git@bastion.em2c.ecp.fr:Test.git myRepo2
```

- Existing projects at EM2C:

```
$ git clone git@bastion:AVBP.git
$ git clone git@bastion:YALES2.git
$ git clone git@bastion:YWC.git
$ git clone git@bastion:REGATH.git
$ git clone git@bastion:CommComb.git
$ git clone git@bastion:DET01D.git
$ git clone git@bastion:UQ.git
$ git clone git@bastion:Rainier.git
.....
```

FILE STATUS LIFECYCLE



MODIFY A GIT REPOSITORY

- Git status (-s)

```
$ touch file1.txt
$ vi file2.txt
$ git status -s
?? file1.txt
?? file2.txt
```

- Git add : stage files to be committed

```
$ git add file1.txt
```

Add file1.txt ony

```
$ git add .
```

Add all files <u>recursively</u>

```
$ git add *
```

Add all files in the current directory <u>only</u>

- Git commit : save local modifications

```
$ git status -s
A file1.txt
A file2.txt
$ git commit -m “add new files”
$ git status -s
$
```

For longer messages:

```
$ git commit
```

Stage modified/deleted files and commit them:

```
$ git commit -a -m “add new files”
```

GIT LOG/DIFF

• Git log (--oneline) : check history

```
$ vi file1.txt
$ git commit -a -m “modify file1”
$ vi file2.txt
$ git commit -a -m “modify file2”
$ git log --oneline
9e7185f modify file2
fd44feb modify file1
bccb3e0 add new files
```

identification key

• Git diff

```
$ vi file2.txt
$ git diff file2.txt
diff --git a/file2.txt b/file2.txt
index e601b6a..74c32cd 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1,2 +1,3 @@
 vflbn
 vfldn
+new line
```

All files:

```
$ git diff
```

Specific tool:

```
$ git difftool -t tkdiff
```

Diff with older commit:

```
$ git diff fd44feb
```

```
$ git diff fd44feb 9e7185f
```

```
$ git diff fd44feb 9e7185f
```

```
$ git diff HEAD 9e7185f
```

IGNORING FILES

- Create a file .gitignore

```
$ git status -s
?? .DS_Store
?? program.o
?? report.pdf
```

→ I don’t want to save this files, never !

All file patterns that must be ignored are specified in the .gitignore file:

```
# No .o files
*.o

# I can save pdf files but not this one:
report.pdf

# Hidden system files
.DS_Store

# Directory tests with one exception
tests/*
!launch_tests.sh
```

→ Comments with #

→ Matches are looked for in directory tree

→ Exceptions with !

You can have different/complementary .gitignore files in subdirectories

Git ignore empty directories

Fix: add a hidden file in the directory (Ex: .gitme), or even a .gitignore file

UNDOING IN GIT

- Git rm/mv

- Un-stage staged files : git reset

```
$ vi file2.txt
$ vi file1.txt
$ vi file3.txt
$ git add .
$ git status -s
M  file1.txt
M  file2.txt
A  file3.txt
```

Oops, I only want to commit file3.txt !

```
$ git reset HEAD file1.txt file2.txt
$ git status -s
 M file1.txt
 M file2.txt
A  file3.txt
$ git commit -m “add file3.txt”
```

All Files:

```
$ git reset HEAD
```

- Fixing un-commited mistakes

Delete all modifications:

```
$ git status -s
 M file1.txt
 M file2.txt
$ git reset --hard HEAD
$ git status -s
$
```

Delete modifications in one file:

```
$ git checkout HEAD file1.txt
$ git status -s
 M file2.txt
$
```

- Fixing commited mistakes

```
$ git revert HEAD
```

```
$ git revert HEAD^
```

```
$ git revert 9e7185f
```

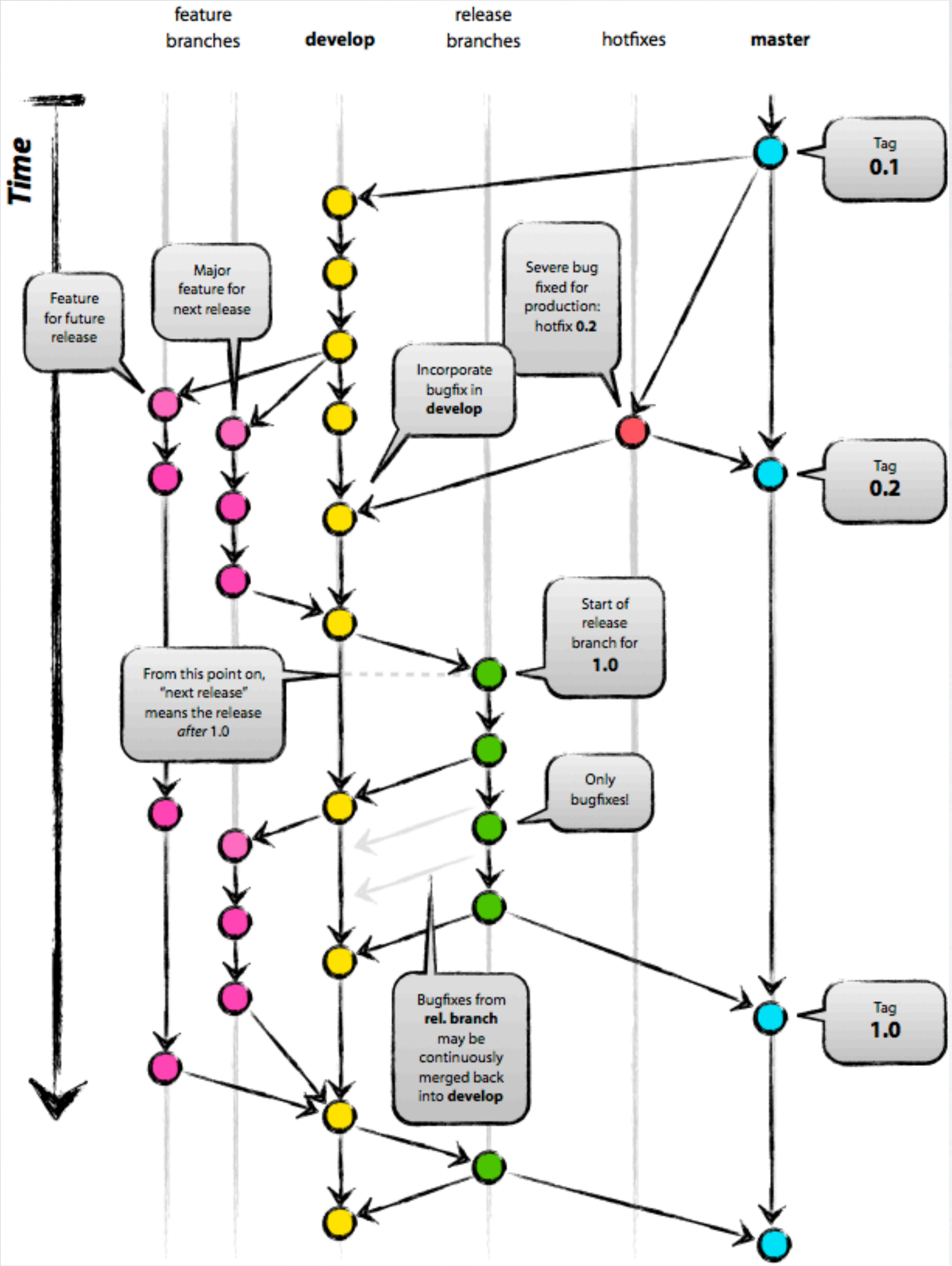
OUTLINE

- **Basics** (git status / add / commit)

- **Branching** (git branch / checkout / merge)

- **Sharing** (git push / fetch / pull)

BRANCHES IN GIT


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

- **Main branches**
  - master
  - develop
- **Feature branches**
  - tend to become user branches on small projects
- **Other branches**
  - hotfixes
  - release

CREATE BRANCHES

- Git branch (-a) : list branches

```
$ git branch
* master
```

- Git branch <name> : create a new branch
  Git checkout <name> : switch to another branch

```
$ git branch develop
$ git branch
develop
* master
$ git checkout develop
$ git branch
* develop
master
$ vi file1.txt
$ git commit -a -m “modify file1”
```

→ I am still in master

→ Now I am in develop

Create and Switch at the same time:

```
$ git checkout -b develop
```

Switch to an older commit:

```
$ git checkout 9e7185f
```

Delete branch:

```
$ git branch -d develop
```

Differences between branches:

```
$ git diff master develop
```

```
$ git diff master develop file1.txt
```

ADD BRANCH IN PROMPT

- Add this in .bashrc

```
# --------------------------------------------------------------------------\
# Setting prompt with user@host path(relative to HOME) and
# git branch if available). With colors.
# --------------------------------------------------------------------------|
# git-track
function parse_git_branch {
  git branch --no-color 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/\1/'
}
function git-track {
  CURRENT_BRANCH=$(parse_git_branch)
  git-config branch.$CURRENT_BRANCH.remote $1
  git-config branch.$CURRENT_BRANCH.merge refs/heads/$CURRENT_BRANCH
}
function parse_git_branch_and_add_brackets {
  git branch --no-color 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/\ \[\1\]/'
}
PS1="\[\e[31;1m\]\u@\[\e[36;1m\]\h: \[\033[0;32m\]\$(parse_git_branch_and_add_brackets) \[\033[0m\] \[\e[32;1m\]\w$ \[\e[0m\]"
# --------------------------------------------------------------------------/
```

Get this in Test.git

```
$ git clone git@bastion:Test.git Tmp
$ cd Tmp
$ vi add_to_bashrc
```

- Automatic display of the current branch in git repositories

```
Ronan@mac ~ $ cd myRepo
Ronan@mac [master] ~/myRepo $ git checkout develop
Ronan@mac [develop] ~/myRepo $
```

MERGE BRANCHES

• Git merge

```
$ git branch
* develop
master
$ git checkout -b NewFancyFeature
$ vi file1.txt
$ git commit -a -m “new amazing code”
$ git checkout develop
$ git branch
* develop
NewFancyFeature
master
$ git merge NewFancyFeature
$ git branch -d NewFancyFeature
```

Create new branch

Merge new branch in develop

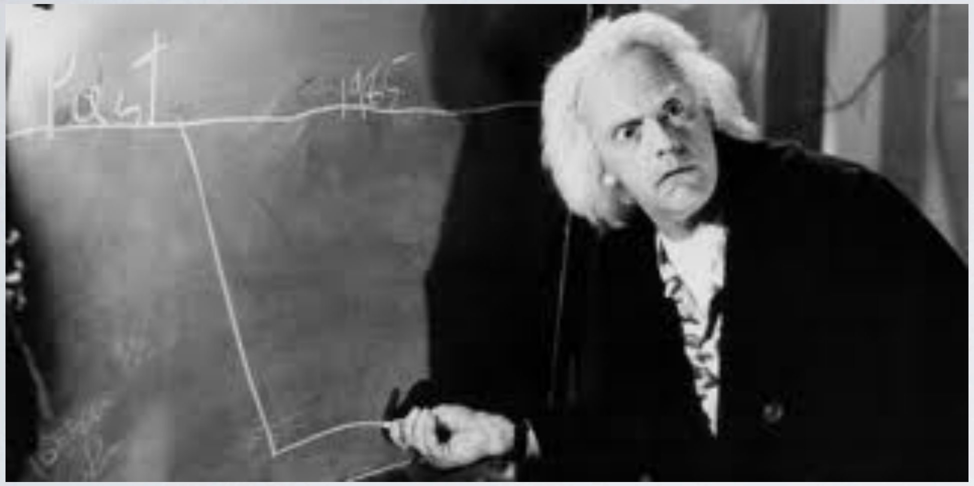
• Merge conflicts

```
$ git merge NewFancyFeature
Auto-merging file3.txt
CONFLICT (content): Merge conflict in file3.txt
Automatic merge failed; fix conflicts and then commit the result.
$ git status -s
UU file3.txt
$ cat file3.txt
<<<<<<< HEAD
Many Hello World Examples
=======
Hello World Lang Examples
>>>>>>> develop
$ git add .
$ git status-s
M file3.txt
```

Specific tool:

```
$ git mergetool -t tkdiff
```

GO BACK IN TIME



You **MUST NOT** change the timeline of the GIT repository
~~CAN NOT~~

(GIT is permissive enough to allow it)

Instead: go back in time and create a new "timeline", ie a branch.

- "Git log" to find the point in time

```
Ronan@mac [feature1] ~/myRepo $ git log --oneline
9e7185f still not working
9e7185f new modifs
fd44feb test new feature
bccb3e0 add new feature
```

This is where I want to start over

- "Git checkout" to go back in time

```
Ronan@mac [feature1] ~/myRepo $ git checkout fd44feb
...
You are in 'detached HEAD' state.
...
Ronan@mac [(no branch)] ~/myRepo $
```

I am nowhere !

- "Git checkout -b" to create a new branch

```
Ronan@mac [(no branch)] ~/myRepo $ git checkout -b feature1-bis
Ronan@mac [feature1-bis] ~/myRepo $
```

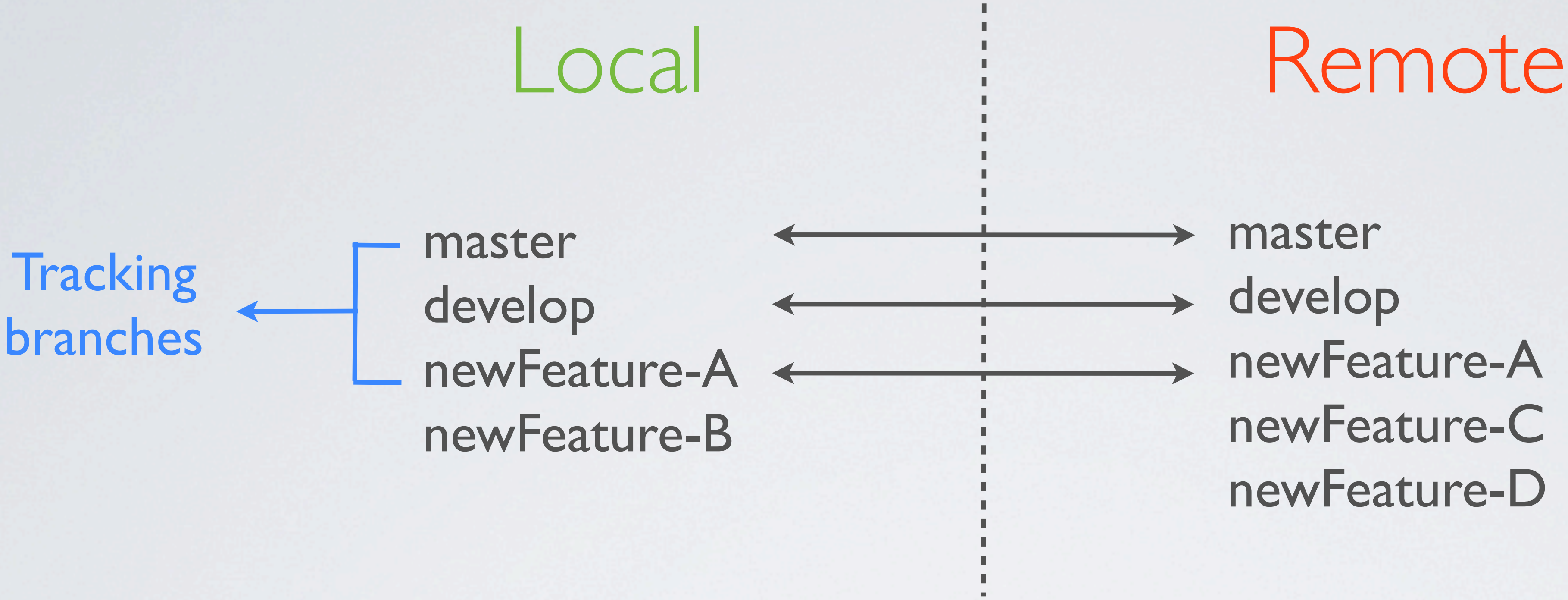
OUTLINE

- **Basics** (git status / add / commit)

- **Branching** (git branch / checkout / merge)

- **Sharing** (git push / fetch / pull)

REMOTE BRANCHES



Local

Remote

Tracking branches

master
develop
newFeature-A
newFeature-B

master
develop
newFeature-A
newFeature-C
newFeature-D

- Git branch -a : list all branches

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/titi
```

- Add remote branches

```
$ git checkout titi
```

```
$ git checkout -t remotes/origin/titi
```

Which branches are tracked ?

```
$ git remote show origin
```

GIT PUSH/PULL/FETCH

- Git push : send modifications to server

Tracking branches: *branch already exists on server*

```
$ git push
```

Create branch on server and track it:

```
$ git push -u origin myBranch
```

Delete branch on server:

```
$ git push origin :myBranch
```

Other local branches: *create new branch on server*

```
$ git push origin myBranch
```

Create branch locally and on server, then track it:

```
$ git branch --track myBranch origin/myBranch
```

- Git fetch : update remote branches

```
$ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
  remotes/origin/titi
```

```
$ git fetch
```

```
$ git fetch origin
```

- Git pull : merge local tracking branches with remote ones

Only for tracking branches:

```
$ git pull
```

Identical to:

```
$ git fetch origin
$ git merge remotes/origin/master
```

CONFLICTS

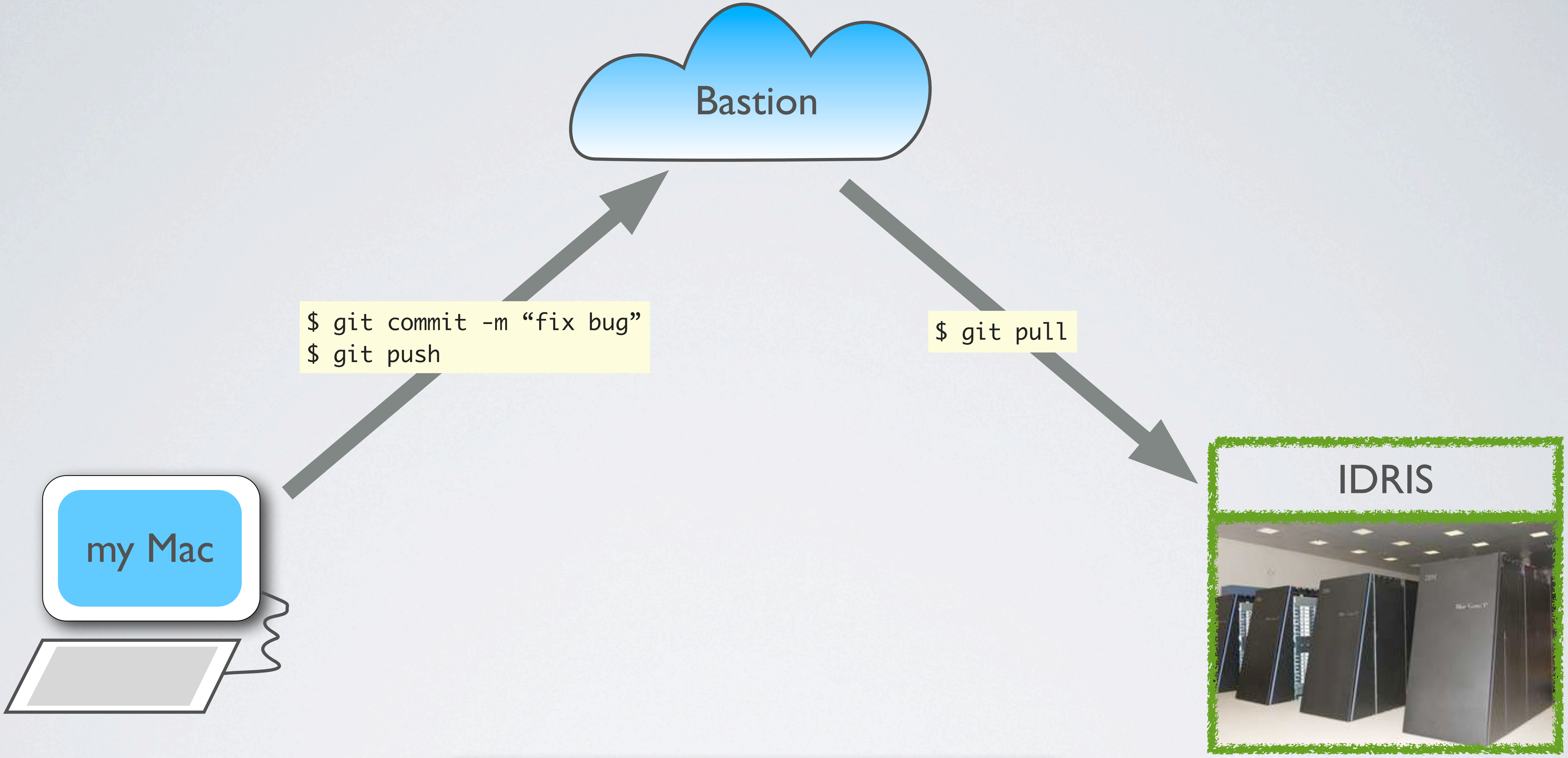
- Conflicts with git pull

You try to pull and get a conflict = merge conflict
You have to fix a usual merge conflict

- Conflicts with git push

Someone else modified the server branch before you.
=> Git won’l let you push your modifications
You have to pull the latest modifications first

PUSH/PULL IGLOO/IDRIS



```
$ git commit -m “fix bug”
$ git push
```

```
$ git pull
```

Try it !

Note : adding remote hosts not shown here on purpose

MISCELLANEOUS

- GUI for Git: SmartGit, GitX ...

- Ignore files: *.o for example
  - Edit .gitignore file
  - Build your code in a different directory

- Gitosis, Gitolite
  - Enable to set access authorization to git repositories on ssh servers
  - Easy to set up (done on bastion)
  - If you need to create a new project or add a new machine you want to connect from : see with the administrator

- Mercurial
  - Mercurial is similar to Git
  - Git = MacGyver
  - Mercurial = James Bond

http://importantshock.wordpress.com/2008/08/07/git-vs-mercurial/

CONCLUSION

90% of the time : status / commit / push / pull

8% of the time : checkout / merge

2% : other

REFERENCES

http://book.git-scm.com/index.html
http://git-scm.com/documentation
http://nvie.com/posts/a-successful-git-branching-model/
http://gitref.org/index.html